In the Specification

Please amend the specification of this application as follows:

Rewrite the paragraph at page 1, lines 18 to 22 as follows:

A2

--This application claims priority under 35 USC §119(e)(1) of Provisional Application No. 60/165,512, filed November 15, 1999 ~~(TI-29909PS)~~, of Provisional Application No. 60/183,527, filed February 18, 2000 ~~(TI-30302PS)~~, and of Provisional Application No. 60/183,609, filed February 18, 2000 ~~(TI-30559PS)~~.--

Rewrite the paragraph at page 4, lines 7 to 10 as follows:

A3

--The increasing demands of technology and the marketplace make desirable even further structural and process improvements in processing devices, application systems, and methods of operation and manufacture.--

Rewrite the paragraph at page 7, line 16 to page 8, line 5 as follows:

A4

--In microprocessor 1 there are shown a central processing unit (CPU) 10, data memory 22, program memory 23, peripherals 60 and an external memory interface (EMIF) with a direct memory access (DMA) 61. CPU 10 further has an instruction fetch/decode unit 10a-c , a plurality of execution units, including an arithmetic and load/store unit D1, a multiplier M1, an ALU/shifter unit S1, an arithmetic logic unit ("ALU") L1, a shared multi-port register file 20a from which data are read and to which data are written. Decoded instructions are provided from the instruction fetch/decode unit 10a-c to the functional units D1, M1, S1, and L1 over various sets of control lines which are not shown. Data are provided to/from the register file 20a from/to to load/store ~~units~~ unit D1 over a first set of busses 32a, to multiplier M1 over a second set of busses 34a, to ALU/shifter unit S1 over a third set of busses

A4

36a and to ALU L1 over a fourth set of busses 38a. Data are provided to/from the memory 22 from/to the load/store ~~units~~ unit D1 via a fifth set of busses 40a. Note that the entire data path described above is duplicated with register file 20b and execution units D2, M2, S2, and L2. Load/store unit D2 similarly interfaces with memory 22 via a set of busses 40b. Instructions are fetched by fetch unit 10a from instruction memory 23 over a set of busses 41. Emulation circuitry 50 provides access to the internal operation of integrated circuit 1 which can be controlled by an external test/development system (XDS) 51. Peripherals 60 connect to external peripherals 80 via bus 80. CPU 10 also includes interrupt controller 90, control logic 100 and configuration registers 102.--

Rewrite the paragraph at page 8, lines 19 to 28 as follows:

A5

--When microprocessor 1 is incorporated in a data processing system, additional memory or peripherals may be connected to microprocessor 1, as illustrated in Figure 1. For example, Random Access Memory (RAM) 70, a Read Only Memory (ROM) 71 and a Disk 72 are shown connected via an external bus 73. Bus 73 is connected to the External Memory Interface (EMIF) which is part of functional block 61 within microprocessor 42. A Direct Memory Access (DMA) controller is also included within block 61. The DMA controller part of functional block 61 connects to data memory 22 via bus 43 and is generally used to move data between memory and peripherals within microprocessor 1 and memory and peripherals which are external to microprocessor 1.--

Rewrite the paragraph at page 9, lines 1 to 3 as follows:

A6

--A detailed description of various architectural features of the microprocessor of Figure 1 is provided in coassigned

A6

~~application S.N. 09/012,813 (TI-25311)~~ U.S. Patent No. 6,182,203 and is incorporated herein by reference.--

Rewrite the paragraph at page 9, lines 4 to 12 as follows:

A7

--Figure 2 is a block diagram of the execution units and register files of the microprocessor of Figure 1 and shows a more detailed view of the buses connecting the various functional blocks. In this figure, all data busses are 32 bits wide, unless otherwise noted. There are two general-purpose register files (A and B) in the processor's data paths. Each of these files contains 32 32-bit registers (A0-A31 for register file A 20a and B0-B31 for register file B 20b). The general-purpose registers can be used for data, data address pointers, or condition registers. Any number of reads of a given register can be performed in a given cycle.--

Rewrite the paragraph at page 11, line 3 to page 12, line 7 as follows:

A8

--Most data lines in the CPU support 32-bit operands, and some support long (40-bit) and double word (64-bit) operands. Each functional unit has its own 32-bit write port into a general-purpose register file (Refer to Figure ~~2A~~ 2). All units ending in 1 (for example, .L1) write to register file A 20a and all units ending in 2 write to register file B 20b. Each functional unit has two 32-bit read ports for source operands *src1* and *src2*. Four units (.L1, .L2, .S1, and .S2) have an extra 8-bit-wide port for 40-bit long writes, as well as an 8-bit input for 40-bit long reads. Because each unit has its own 32-bit write port, when performing 32-bit operations all eight units can be used in parallel every cycle. Since each multiplier can return up to a 64-bit result, two write ports (dst1 and dst2) are provided from the multipliers to the respective register file.--

Rewrite the paragraph at page 12, lines 10 to 18 as follows:

A9

--Each functional unit reads directly from and writes directly to the register file within its own data path. That is, the .L1 unit 18a, .S1 unit 16a, .D1 unit 12a, and .M1 ~~units~~ unit 14a write to register file A 20a and the .L2 unit 18b, .S2 unit 16b, .D2 unit 12b, and .M2 ~~units~~ unit 14b write to register file B 20b. The register files are connected to the opposite-side register file's functional units via the 1X and 2X cross paths. These cross paths allow functional units from one data path to access a 32-bit operand from the opposite side's register file. The 1X cross path allows data path A's functional units to read their source from register file B. Similarly, the 2X cross path allows data path B's functional units to read their source from register file A.--

Rewrite the paragraph at page 12, lines 19 to 23 as follows:

A10

--All eight of the functional units have access to the opposite side's register file via a cross path. The .M1, .M2, .S1, .S2, .D1, and .D2 units' *src2* inputs are selectable between the cross path and the same side register file. In the case of the .L1 and .L2 both *src1* and *src2* inputs are also selectable between the cross path and the same-side register file. Cross path 1X bus 210 couples one input of multiplexer 211 for src1 input of .L1 unit 18a, multiplexer 212 for src2 input of .L1 unit 18a, multiplexer 213 for src2 input of .S1 unit 16a and multiplexer 214 for scr2 input of .M1 unit 14a. Multiplexers 211, 212, 213, and 214 select between the cross path 1X bus 210 and an output of register file A 20a. Buffer 250 buffers cross path 2X output to similar multiplexers for .L2, .S2, .M2, and .D2 units.--

Insert the following paragraph at page 13, between lines 9 to 10:

A11 --S2 unit 16b may write to control register file 102 from its dst output via bus 220. S2 unit 16b may read from control register file 102 to its src2 input via bus 221.--

Rewrite the paragraph at page 13, line 24 to page 14, line 4 as follows:

A12 --Bus 40a has an address bus DA1 which is driven by mux 200a. This allows an address generated by either load/store unit D1 or D2 to provide a memory address for loads or stores for register file 20a. Data Bus LD1 loads data from an address in memory 22 specified by address bus DA1 to a register in load unit D1. Unit D1 may manipulate the data provided prior to storing it in register file 20a. Likewise, data bus ST1 stores data from register file 20a to memory 22. Load/store unit D1 performs the following operations: 32-bit add, subtract, linear and circular address calculations. Load/store unit D2 operates similarly to unit D1 via bus 40b, with the assistance of mux 200b for selecting an address.--

[illegible] Rewrite the paragraph at page 14, lines 18 to 29 as follows:

A13 --Table 3 defines the mapping between instructions and functional units for a set of basic instructions included in DSP 10 is described in U.S. Patent ~~S.N. 09/012,813~~ No. 6,182,203 (~~TI-25311,~~ incorporated herein by reference). Table 4 defines a mapping between instructions and functional units for a set of extended instructions in an embodiment of the present invention. A complete description of the extended instructions is provided in U.S. Patent ~~S.N. ______ (TI-30302)~~ Application Serial No. 09/703,096 entitled "Microprocessor with Improved ISA," and is incorporated herein by reference. Alternative embodiments of the present invention may have different sets of instructions and functional unit mapping. Table 3 and Table 4 are illustrative and are not

A13 exhaustive or intended to limit various embodiments of the present invention.--

Rewrite the paragraph at page 19, lines 17 to 23 as follows:

A14 --Since the pipeline of the present embodiment is unprotected, certain code sequences that cause resource conflicts are invalid. An assembler with appropriate code sequence checking capabilities is used to screen out invalid code sequences. Several constraints related to the present embodiment are described below. Other constraints which may apply to the present embodiment are described in US Patent No. 6,112,298 ~~(TI-24946)~~, incorporated herein by reference.--

Rewrite the paragraph at page 20, lines 20 to 27 as follows:

A15 --The pipeline operation, from a functional point of view, is based on CPU cycles. A CPU cycle is the period during which a particular execute packet is in a particular pipeline stage. CPU cycle boundaries always occur at clock cycle boundaries; however, memory stalls can cause CPU cycles to extend over multiple clock cycles. To understand the machine state at CPU cycle boundaries, one must be concerned only with the execution phases (E1-E5) of the pipeline. The phases of the pipeline are shown in Figure ~~11~~ 5 and described in Table 8.--

Rewrite the paragraph at page 23, lines 21 to 29 as follows:

A16 --Branch instructions execute during the E1 phase of the pipeline five delay slots/CPU cycles after the branch instruction enters an initial E1 phase of the pipeline. ~~Figure 12 shows the branch instruction phases.~~ Figure ~~13~~ 5 shows the operation of the pipeline based on clock cycles and fetch packets. In Figure ~~13~~ 5, if a branch is in fetch packet n, then the E1 phase of the branch is the PG phase of n+6. In cycle 7 n is in the E1 phase and n+6 is

A16 in the PG phase. Because the branch target is in PG on cycle 7, it will not reach E1 until cycle 13. Thus, it appears as if the branch takes six cycles to execute, or has five delay slots.--

Rewrite the paragraph at page 24, line 18 to page 25, line 6 as follows:

A17 --Figure 6 is a block diagram illustrating a prior art processor 600 that requires execution packets to be aligned within fetch packets. Prior art processor 600 is a VLIW RISC architecture that has an instruction execution pipeline that operates similarly to processor 10 in aspects other than execution packet alignment. Four fetch packets ~~611-613~~ 610-613 are illustrated in Figure 6. These are each fetched sequentially in response to program fetch circuitry in processor 600. Fetch packet 610 comprises an execution packet 620 that has seven useful instructions 620(0)-620(6) that can all be executed in parallel. The next execution packet 621 is in fetch packet 611 and comprises instructions 621(0)-621(4). A third execution packet 622 comprises one useful instruction 622(0). Disadvantageously, a no-operation instruction (NOP) 620(7) must be included in execution packet 620 since execution packets must be aligned within fetch packets. Similarly, NOP instructions 622(1) and 622(2) are placed in execution packet 622 to cause alignment with fetch packet 611. NOP instructions 620(7), 622(1) and 622(2) waste storage resources in processor 600. Note that the last word of each fetch packet has the p-bit set to 0 to indicate the end of an execute packet, such as instruction word 620(7) and 622(2), for example.--

Rewrite the paragraph at page 25, line 16 to page 26, line 2 as follows:

A18 --Figure 7B is an illustration of execution packets spanning fetch packets for the processor of Figure 1. Advantageously, in

A18

the present embodiment of processor 10, an execution packet can cross an eight-word fetch packet boundary, thereby eliminating a need to add NOP instructions to pad fetch packets. For example, eight-word execution packet EP1 completely occupies fetch packet 700. Four-word execution packet EP2 partially fills fetch packet 702. Six-word execution packet EP3 does not fit completely within fetch packet 702, however, the first four words ~~EP#(0)-EP3(3)~~ EP3(0)-EP3(3) are placed in fetch packet 702 and the last two words EP3(4), EP3(5) are placed in fetch packet 704. Therefore, the last *p*-bit in a fetch packet is not always set to 0 in processor 10. If the last p-bit of a fetch packet is not zero, then instruction fetch control circuitry in stage 10a (Figure 1) fetches a second fetch packet and extracts instruction words until a p-bit set to 0 is encountered. This sequence of instruction words is then ordered into a single execution packet, such as execution packet EP3, for example.--

Rewrite the paragraph at page 26, lines 3 to 19 as follows:

A19

--Figure 8 is a block diagram of processor 10 of Figure 1, illustrating a sequence of execution packets spanning fetch packets 810-813, according to an aspect of the present invention. For ease of comparison, the same program portion is illustrated here as in Figure 6. For example, execution packet 820 comprises only the seven useful instructions 820(0)-820(6). The last instruction word 820(6) has the p-bit set to 0. Advantageously, the first word 821(0) of execution packet 821 is now placed in fetch packet 810. Note that the p-bit of instruction word 821(0) is set to 1 to indicate that execute packet 821 continues to the next fetch packet 811 where instruction words 821(1)-821(4) are located. Likewise, execution packet 824 spans fetch packets 811 and 812 with instruction word 824(0) located in fetch packet 811 and instruction words 824(1)-824(3) located in fetch packet 812. Execution packet

822 with instruction word 822(0) is located within fetch packet 811. Execution packet 823 with instruction words 823(0)-823(1) is located within fetch packet 811. Note that the last word of the fetch packet now does not necessarily have the p-bit set to 0, for example instruction words 821(0) and 824(0) both have their p-bit set to one indicating that their associated execution packet spans to the next fetch packet.--

A19

Rewrite the paragraph at page 28, lines 21 to 24 as follows:

A20

--Three multi-channel buffered serial ports (McBSP) 1060, 1062, 1064 are connected to DMA controller 1040. A detailed description of a McBSP is provided in U.S. Patent ~~application S.N. 09/055,011 (TI-26204, Seshan, et al)~~ No. 6,167,466 and is incorporated herein reference.--

Rewrite the paragraph at page 29, line 23 to page 30, line 5 as follows:

A21

--Figure 11 illustrates an exemplary implementation of an example of an integrated circuit 40 that includes digital system 1000 in a mobile telecommunications device, such as a wireless telephone 15 with integrated keyboard 12 and display 14. As shown in Figure 11 digital system 1000 with processor ~~1001~~ 10 (co-located with integrated circuit 40) is connected to the keyboard 12, where appropriate via a keyboard adapter (not shown), to the display 14, where appropriate via a display adapter (not shown) and to radio frequency (RF) circuitry 16. The RF circuitry 16 is connected to an aerial 18. Advantageously, by allowing execute packets to span fetch packets, memory is not wasted with useless NOP instructions. Thus, a smaller memory can be included within the wireless telephone and fewer fetch cycles are required for execution of a given processing algorithm and power consumption is thereby reduced.--